データマイニングによる プログラム中のイディオムの発見

田中 哲 <akr@m17n.org>

産業技術総合研究所 情報処理研究部門

目的

ライブラリを使いやすく改善する

方針:

1. イディオムを発見
2. イディオムを直接実現する API を追加

イディオム

プログラム中に良く現われる慣用句
C のイディオムの例:

```
while (*p++ = *q++) ;
```

デザインパターンよりは小さなものを指す

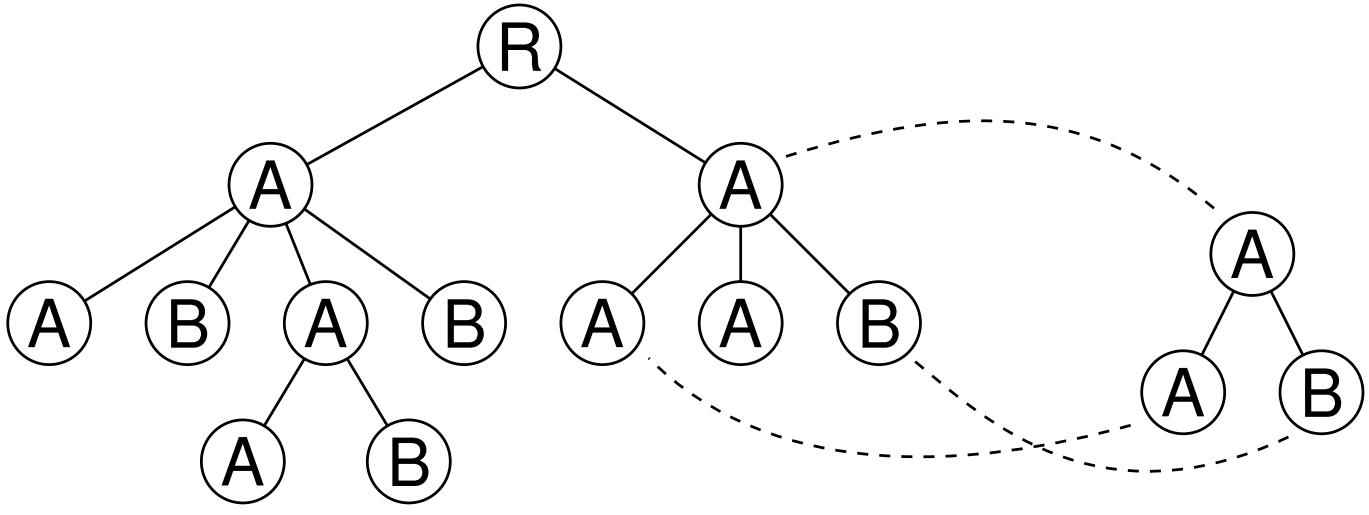
データマイニング

大量のデータから半自動的に規則性を見つけ出す方法
いろいろなアルゴリズムがある

- 半構造データ (XML) を対象とするアルゴリズム: FREQT
- リレーショナルデータベースを対象とするアルゴリズムなど

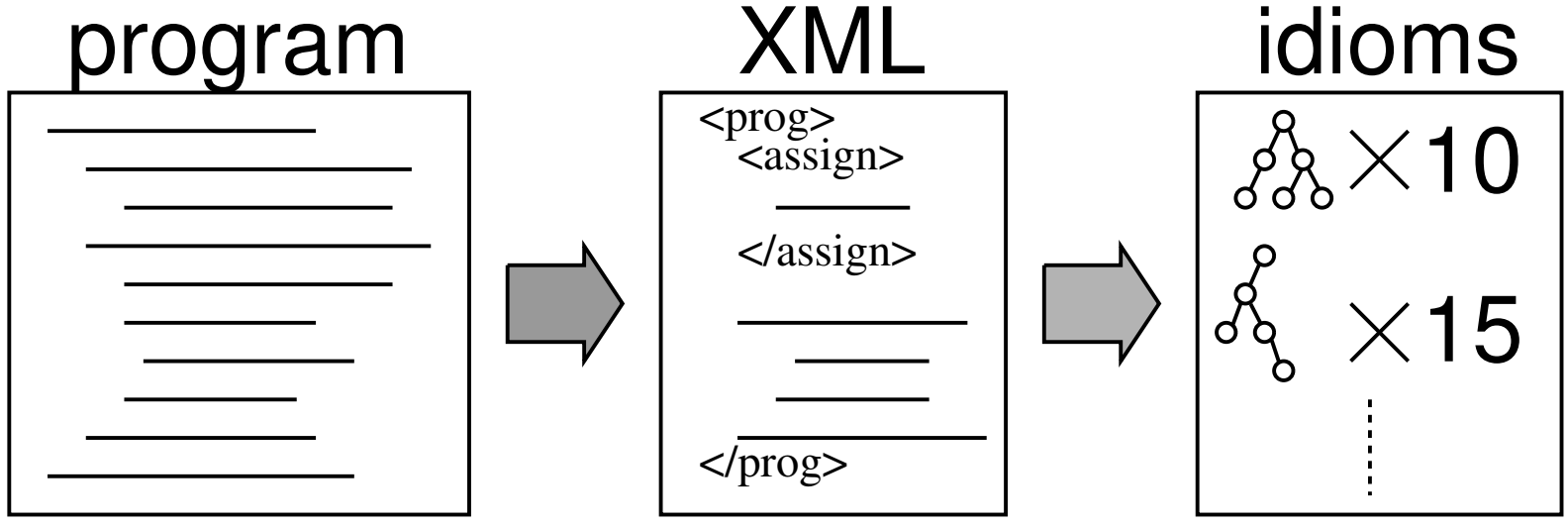
FREQT

半構造データ (XML) を対象とするデータマイニングアルゴリズム [浅井 et al, 2002]
ラベルつき順序木の中から瀕出パターンを発見する



イディオムの発見

1. プログラムを構文木に変換して XML で表現
2. XML に FREQT を適用



ライブラリの改善

検出したイディオムに対応する API を追加

obj.m1() + obj.m2() が瀕出

↓

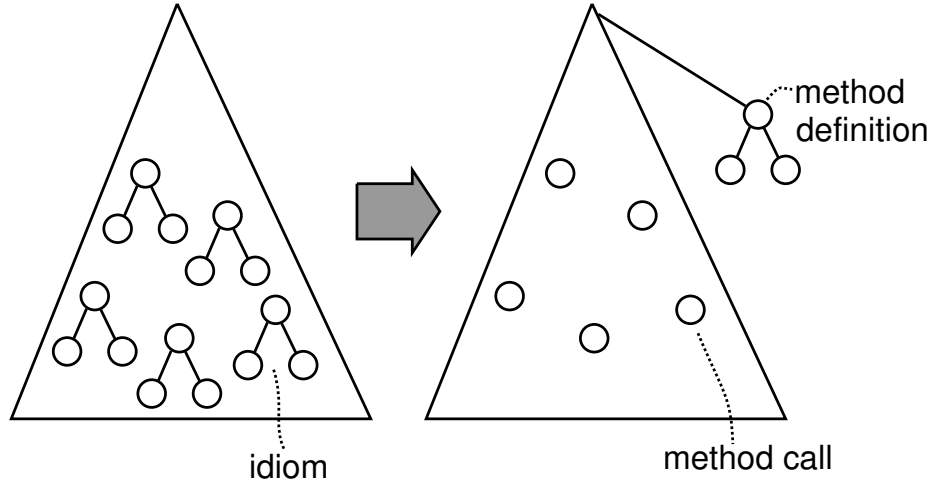
obj.m3() と書けるように m3 を定義

頻繁に行うことを簡単に書けるようになる

改善の度合

コードがどの程度短くなるか
obj.m1() + obj.m2() → obj.m3()

例: 7ノード減:



パターンの大きさと出現頻度の積に比例

Ruby のライブラリで検出の実験

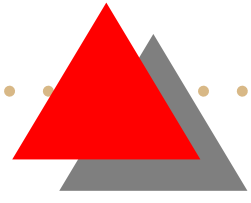
English.rb 「alias 変数 変数」 × 27

base64.rb 「式.gsub!(regexp)」 × 4
「変数.gsub!(regexp) { decode64(変数) }」
× 2

benchmark.rb
「変数.gsub!(regexp) { 文字列 % 変数 }」
× 7

検出の工夫

- 不要な構造の削除: クラス、メソッド定義は body を展開
- 引数位置の扱い



引数位置の扱い

FREQT では子の出現位置を考慮しないので、メソッド呼び出しの引数など、位置に意味がある場合は工夫が必要
m(1,2) と m(2,3) から、m(2) が見つかってしまう
次のようにエンコードしてある程度扱えるようにした

```
<m_methodname>
  <arg1>...</arg1>
  <arg2>...</arg2>
</m_methodname>
```

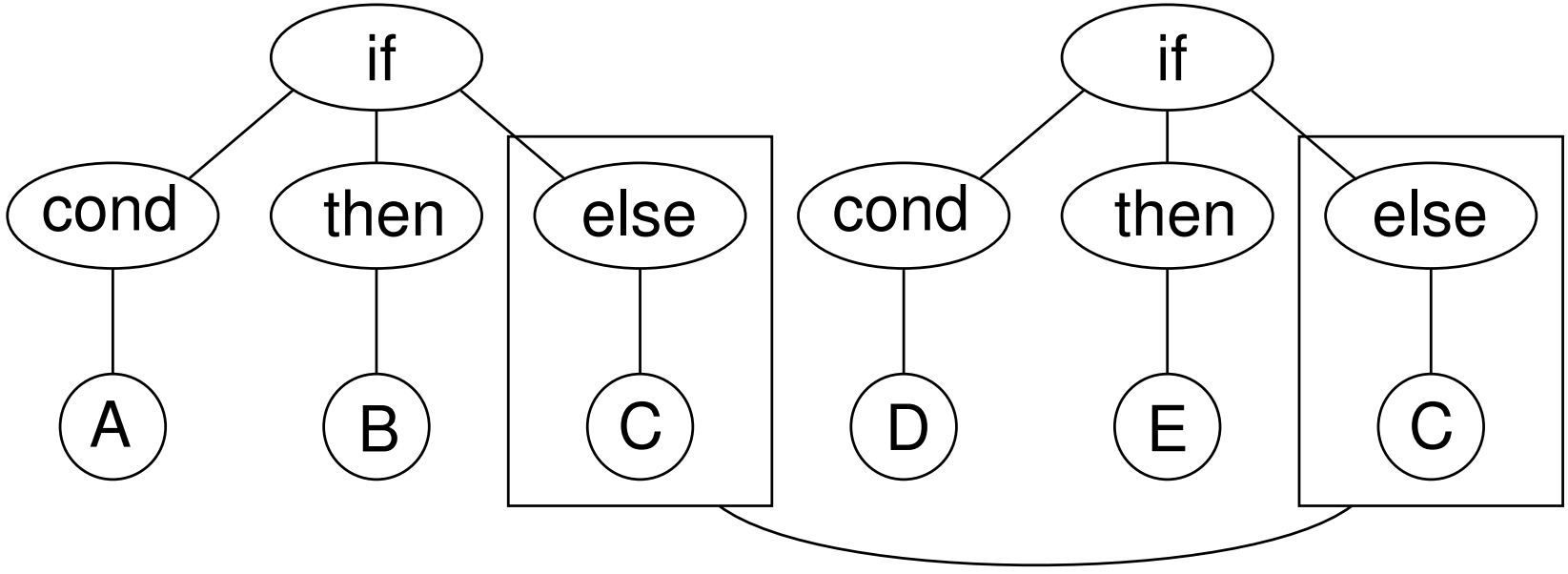
検討事項

- メソッドとして切り出せる単位のパターンだけを検出する
- 興味がある場所の周辺だけを対象にする
- 順序の無い構造の扱い
- 名前の一致の扱い
- etc.

検討事項: メソッド化可能なパターンのみを検出

式としての要素をルートとするパターンだけを探す

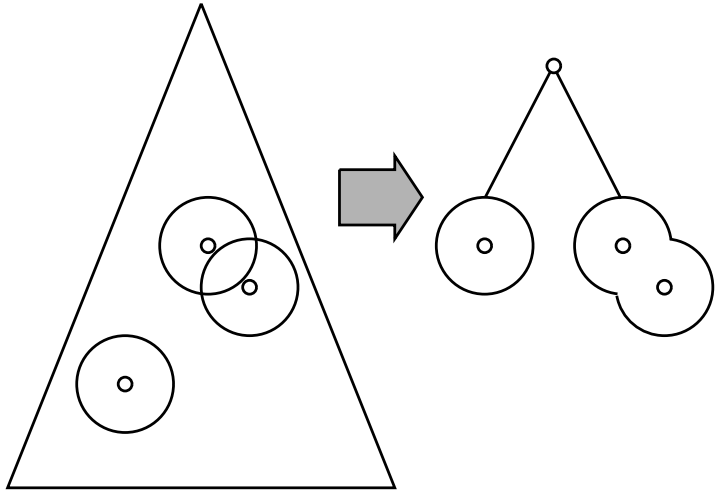
例: else 節をルートとするパターンは無視



メソッドには切り出せない

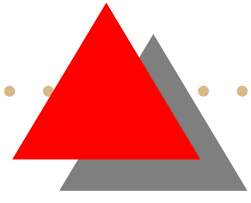
検討事項: 興味がある場所の周辺だけを対象にする

ライブラリが提供するメソッドを使用している周辺だけを取り出してからFREQTに適用する



検討事項: 順序の無い構造の扱い

クラスの集合からのパターンの発見
デザインパターンなど



検討事項: 名前の一致の扱い

α 変換したようなものを発見できるか?
現在は名前の違いは無視

`t = a`	変数 = 変数
`a = b`	変数 = 変数
`b = t`	変数 = 変数

とりあえず見つけたいもの

Java:

```
for (Iterator iter = ...;
     iter.hasNext();) {
  Object o = iter.next();
  ...
}
```

Ruby:

```
/...#{式.join('|')}.../
```

無理なこと

もともとできなかったことをできるようにする改善は無理

将来の予定

- AST の改良: ライブラリの改良のためのデータマイニングに適した AST を設計する
- データフローグラフに対するマイニング
- XML 関係のライブラリの比較

まとめ

- データマイニングでプログラム中から瀕出パターンを抽出した
- いくつか idiom らしきものが見つかった